

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 2003

Quiz I

Closed Book – one sheet of notes

Separately, we have distributed an answer sheet. You may use the space on the exam booklet for whatever temporary work you find useful, but you **MUST** enter your answers into the answer sheet. **Only the answer sheet will be graded.** Each problem that requires an answer has been numbered. Place your answer at the corresponding number in the answer sheet.

Note that any procedures or code fragments that you write will be judged not only on correct function, but also on clarity and good programming practice.

The answer sheet asks for your section number and tutor. For your reference, here is the table of section numbers.

Section	Time	Location	Rec. Instructor	Tutors
1	10:00	26-328	Joel Moses	Bryan Russell
2	10:00	36-155	Regina Barzilay	Ben Vandiver
3	11:00	26-328	Joel Moses	Dave Feinberg
4	12:00	36-144	Regina Barzilay	
5	1:00	26-204	Randy Davis	Limor Fried
6	2:00	26-204	Randy Davis	Vivienne Lee

Part 1: (16 points)

For each of the following expressions or sequences of expressions, state the value returned as the result of evaluating the final expression in each set, or indicate that the evaluation results in an error. If the expression does not result in an error, also state the “type” of the returned value, using the notation from lecture. If the result is an error, state in general terms what kind of error (e.g. you might write “error: wrong type of argument to procedure”). If the evaluation returns a built-in procedure, write **primitive procedure**, and its **type**. If the evaluation returns a user-created procedure, write **compound procedure**, and also indicate its **type** using the notation we introduced in class. You may assume that evaluation of each sequence takes place in a newly initialized Scheme system.

Question 1.

```
(+ 2 (* 3 4))
```

Question 2.

```
(let ((* /)
      (/ *))
      (/ 12 (* 6 3)))
```

Question 3.

```
((lambda (x + y) (+ x y))
 3 * 4)
```

Question 4.

```
(lambda (x y)
  (if (> x 0) x (sqrt y)))
```

Question 5.

```
(lambda (a b)
  (if (> a 0)
      b
      (lambda (x) (b (- x)))))
```

Question 6.

```
(define (square x) (* x x))
(define (foo f n)
  (if (= n 0)
      (lambda (x) x)
      (lambda (x)
        (f ((foo f (- n 1)) x)))))

((foo square 2) 3)
```

Now that it is October, many fans' attention turns to baseball's World Series. Boston Red Sox fans are used to seeing their team find inventive ways to lose in the playoffs, or even fail to reach them. We are going to see if we can provide some insight into the best possible lineup for the team, by building a database of statistics (if you don't know much about baseball, don't worry; the questions should be self-explanatory).

Here is a simple data abstraction for the database. We will represent a roster (or set of players) as a list. Each entry in the roster will initially include information about the player, his times at bat, and the number of hits in those at bats.

```
(define (make-name last first) (list last first))

(define last-name first)
(define first-name second)

(define (make-roster-entry last first hits atbats)
  (list (make-name last first) hits atbats))

(define player-names first)
(define player-hits second)
(define player-atbats third)
```

Here is a sample database for the Red Sox

```
(define bosox
  (list (make-roster-entry "garciaparra" "nomar" 195 630)
        (make-roster-entry "damon" "johnny" 161 584)
        (make-roster-entry "ramirez" "manny" 176 544)
        ...
        (make-roster-entry "varitek" "jason" 117 426)
        (make-roster-entry "ortiz" "david" 120 424)))
```

Note that we are using strings to represent the names of players, both first and last name.

You may find the following definitions useful:

```
(define (map proc lst)
  (if (null? lst)
      nil
      (cons (proc (car lst))
            (map proc (cdr lst)))))

(define (filter pred lst)
  (cond ((null? lst) nil)
        ((pred (car lst)) (cons (car lst) (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))

(define (fold-right op init lst)
  (if (null? lst)
      init
      (op (car lst) (fold-right op init (cdr lst)))))
```

Part 2: (24 points)

We would like to add information about a player's batting average (the percentage of atbats in which a player gets a hit). Traditionally, we multiply this fraction by 1000, and round off (e.g. using Scheme's `round` procedure) to get a number ranging from 0 to 1000.

For example, we would like the following behavior:

```
(first bosox)
;Value: (("garciaparra" "nomar") 195 630)

(player-average (first bosox))
;Value: 310.
;; Note that 1000 * 195/630, rounded off is 310
```

Question 7. Let's revise the roster-entry abstraction to compute the player's average (computed, not input). This means we should be able to use `player-average` to access any entry in the roster, and return the batting average for that player, even though it was not provided when we first created the entry. For each of the following procedures, write a new version to reflect this change, or write "no change" if nothing needs to change.

```
make-roster-entry
player-name
player-hits
player-atbats
```

Question 8. Complete the following definition:

```
(define (player-average entry)
  Q8-ANSWER)
```

Now, the manager is trying to use the roster abstraction, including the new capabilities we added, to pick players for the lineup. He may want to extract from the roster the entry for a particular player, for example, by last name:

```
(find-player bosox "varitek")
;Value: (("varitek" "jason") 117 426)
```

Question 9. First, provide a selector that operates on a roster entry, and returns the last name of that player (be sure to avoid abstraction violations!), e.g.

```
(player-last-name (first bosox))
;Value: "garciaparra"

(define (player-last-name entry)
  Q9-ANSWER)
```

Now complete the definition for `find-player` (note that `string=?` can be used to compare two strings for equality):

```
(define (find-player roster last-name)
  (cond ((null? roster) "not present")
        (Q10-ANSWER (first roster))
        (else Q11-ANSWER)))
```

Question 10. What code should be inserted in place of Q10-ANSWER?

Question 11. What code should be inserted in place of Q11-ANSWER?

Question 12. Suppose the manager wants to create a list of all the players, combining last name and batting average:

```
(get-average-stats bosox)
;Value:
(("garciaparra" 310.)
 ("damon" 276.)
 ("walker" 282.)
 ("ramirez" 324.)
 ("millar" 272.)
 ("meuller" 331.)
 ("nixon" 309.)
 ("varitek" 275.)
 ("ortiz" 283.))
```

Complete the following definition:

```
(define (get-average-stats roster)
  (map Q12-ANSWER roster))
```

Part 3: (34 points)

Suppose we want to get information about statistics in the database, for example, a list of the batting averages of the players, **sorted** in decreasing order:

```
(sort-stats bosox player-average)
;Value: (331. 324. 310. 309. 283. 282. 276. 275. 272.)

(sort-stats bosox player-hits)
;Value: (195 176 166 161 158 140 133 120 117)
```

To sort the data, we will first extract a list of the relevant data, and then sort that list:

```
(define (sort-stats roster category)
  (let ((stats (map category roster)))
    (sort-lst-decrease stats)))
```

You need to help us write `sort-lst-decrease`. Here is the framework:

```
(define (sort-lst-decrease lst)
  (if (null? lst)
      nil
      (let ((best (find-best lst)))
        (cons best (sort-lst-decrease (remove best lst))))))
```

Question 13. The procedure `remove` should return a version of its second argument, with all instances of its first argument removed from it. Assume that all the elements are numbers. Complete the following definition (please do not use `delq`):

```
(define (remove val lst)
  (filter Q13-ANSWER lst))
```

The procedure `find-best` should return the best value from the list, determined here to be the largest value. You may assume that all entries are greater than or equal to 0. Complete the following definition:

```
(define (find-best lst)
  (fold-right Q14-ANSWER Q15-ANSWER lst))
```

Question 14. What code should be used in place of Q14-ANSWER?

Question 15. What code should be used in place of Q15-ANSWER?

So far, we have created the ability to sort lists of numbers. But suppose we wanted to sort the entire data structure, for example, by last name of the player (using `string<=?` as predicate for ordering strings):

```
(sort-roster bosox
  (lambda (x y) (string<=? (player-last-name x) (player-last-name y))))
```

This is a generalization of our previous search method:

```
(define (sort-roster roster compare)
  (if (null? roster)
      nil
      (let ((next (find compare (car roster) (cdr roster))))
        (cons next (sort-roster (remove-full next roster) compare)))))
```

You need to complete this procedure:

Question 16.

Complete the definition for `remove-full`, which should create a new roster, without the specified entry (remember that this is a full roster entry):

```
(define (remove-full elt roster)
  (filter Q16-ANSWER roster))
```

Here is a template for `find`, complete it:

```
(define (find compare best rest)
  (if (null? rest)
      Q17-ANSWER
      (if (compare best (car rest))
          Q18-ANSWER
          Q19-ANSWER)))
```

Question 17. What code is needed for Q17-ANSWER?

Question 18. What code is needed for Q18-ANSWER?

Question 19. What code is needed for Q19-ANSWER?

Now suppose we apply this to our Red Sox roster:

```
(define sorted-bosox
  (sort-roster bosox
    (lambda (x y) (string<? (player-last-name x) (player-last-name y)))))
```

This gives us a sorted list of player entries. Suppose we now want to insert a new player entry into the right spot in this sorted list. Assume that `string<?` is a predicate that applies to two strings, and returns `#t` if the first string is lexicographically smaller than the second (i.e. would appear closer to the front of a dictionary).

```
(define (insert-by-name new roster)
  (if (null? roster)
      Q20-ANSWER
      (if (string<=? (player-last-name (car roster)) (player-last-name new))
          Q21-ANSWER
          Q22-ANSWER)))
```

Question 20. What code is needed for Q20-ANSWER?

Question 21. What code is needed for Q21-ANSWER?

Question 22. What code is needed for Q22-ANSWER?

Question 23. If the roster has size n , what is the order of growth of `insert-by-name` in time? Choose from:

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- other

Question 24. If the roster has size n , what is the order of growth of `insert-by-name` in space (as measured by the number of deferred operations)? Choose from:

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- other

Once we have `insert-by-name` we could actually build a sorted roster from scratch, as follows

```
(define (create-sorted-roster lst sorted-lst)
  (if (null? lst)
      sorted-lst
      (create-sorted-roster (cdr lst) (insert-by-name (car lst) sorted-lst))))

(define sorted-bosox (create-sorted-roster bosox nil))
```

Question 25. If the roster has size n , what is the order of growth of `create-sorted-roster` in time? Don't forget your answer to Question 24. Choose from:

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- other

Question 26. If the roster has size n , what is the order of growth of `create-sorted-roster` in space? Don't forget your answer to Question 25. Choose from:

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- other

Part 4: (16 points)

Fibonacci is a function of a non-negative integer, defined as follows:

$$\begin{aligned}f(0) &= 1 \\f(1) &= 1 \\f(n) &= f(n - 1) + f(n - 2), \text{ if } n \geq 2\end{aligned}$$

Question 27.

Write a procedure to implement Fibonacci that has a recursive order of growth (i.e. $O(n)$ in space).

Question 28.

Write a procedure to implement Fibonacci that has an iterative order of growth (i.e. $O(1)$ in space).

Part 5: (10 points)

Consider the following procedures:

```
(define (arg&fun f)
  (lambda (x) (* x (f x))))
```

```
(define (again f n)
  (if (= n 1)
      f
      (lambda (g) ((again f (- n 1)) (f g)))))
```

```
(define (id x) x)
```

```
(define (more x) (+ x 2))
```

Question 29. What value is returned by the following expression

```
((arg&fun id) 7)
```

Question 30. What value is returned by the following expression

```
((arg&fun more) 3)
```

Question 31. What value is returned by the following expression

```
((again arg&fun 3) id) 2)
```

Question 32. What value is returned by the following expression

```
((again arg&fun 3) more) 2)
```